

# Linear algorithm for data retrieval performance optimization in self-encryption hybrid data centers

Maen M. Al Assaf<sup>1</sup>, Mohammad Qatawneh<sup>1,2</sup>, AlaaAldin AlRadhi<sup>3</sup>

<sup>1</sup>King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan

<sup>2</sup>Al-Ahliyya Amman University, Amman, Jordan

<sup>3</sup>Sheridan College, Toronto, Canada

## Article Info

### Article history:

Received Sep 18, 2024

Revised Jan 2, 2025

Accepted Mar 9, 2025

### Keywords:

Data retrieve

Hybrid storage systems

Linear algorithm

Storage system bandwidth

Symmetric self-encryption

devices

## ABSTRACT

Contemporary data centers implement hybrid storage systems that consist of layers from solid-state drives (SSDs) and hard disk drives (HDDs). Due to their high data retrieval speed, SSDs layer is used to store important data blocks that have features like high frequency of access. To boost their security level, many of such systems implement self-encryption algorithms like advanced encryption standard (AES), Blowfish, and triple data encryption standard (3DES) with different key sizes that vary in their complexity and their decryption latency whenever a block is requested for read. Frequently accessed data blocks with increased decryption latencies are better to be migrated to the SSDs layer to decrease their retrieval latency. In this paper, we introduce a linear complexity algorithm hybrid self-encryption storage data migration (HSESM) that migrates important data blocks that requires long decryption latencies from the HDDs layer to the SSDs one. Performance evaluation shows that HSESM data migration process can reduce data blocks read latencies in 13.71%-23.61% under worst-case scenarios.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



## Corresponding Author:

Maen M. Al Assaf

King Abdullah II School for Information Technology, University of Jordan

Amman 11942, Jordan

Email: m\_alassaf@ju.edu.jo

## 1. INTRODUCTION

Distributed systems in today's world including cloud systems use distributed data centers to store data that increases rapidly in terms of size. Most of data centers are designed in a hybrid approach where they consist of array of hard disk drives (HDDs) and solid-state drives (SSDs) [1]. The SSDs are known of their high data read/write speed performance in comparison to the HDDs, but in the same, their cost per storage unit is high. Hence, data centers will need for HDDs as a lower cost storage to enable the data center to accommodate the huge amount of data [2]. SSDs are used to store data blocks that are continuously needed to be retrieved by users' applications in the read operations, whereas HDDs store less needed data blocks [1].

When it comes to the security and privacy for those data blocks stored in the data centers, symmetric self-encryption devices (SEDs) is implemented on both layers of the hybrid system by using variant symmetric encryption algorithms [3]. The data blocks that are written on or read from either the HDDs or the SSDs layer are encrypted or decrypted using a supplementary processing engine attached with the devices [4]. This boosts the security level for the data stored on the storage system. However, data decryption latency creates extra overhead when a particular data block is requested to be retrieved.

Due to the significant speed difference between the SSDs and the HDDs, SSDs will be the best place to store the continuously required data blocks by the users' applications. So, when these data blocks are

requested, the total read latency time that includes their decryption and reading will be significantly less in comparison to the case if they allocated on the HDDs. For this reason, it is important to continuously assess data blocks stored on the hybrid system using features like their access frequency, if they belong to real-time or foreground applications, and if they use complex encryption algorithms that take longer time in decryption in order to indicate if each particular data block is more important to be read faster. This assessment helps making important data blocks that are highly accessed and take long time in decryption process to be migrated from the HDDs layer to the SSDs one in case if they were not originally located on the SSDs. In fact, the chance of finding that a data block is originally allocated on the HDDs layer is higher due to their significant volume [5]. However, moving the data blocks that are classified as important from the HDDs to the SSDs and those that are classified as less important in the opposite direction may consume the available bandwidth of the hybrid storage system. This is because there exists a maximum number of I/O requests that may take place concurrently in any storage system without being congested [1], [5].

Based on this introduction to our research, it is worthwhile to list its primary motivations that underscore its significance:

- a. Hybrid storage systems are widely used in today's distributed data centers to provide high storage capacity and data retrieve performance.
- b. There exists a significant gap in data read performance between SSDs and HDDs implemented in such systems. Hence, data migration is important to move important data to the SSDs layers that show faster read performance.
- c. Hybrid storage systems implement self-encryption algorithms to boost security level for the stored data. Data decryption latency creates extra overhead when a particular data block is requested to be retrieved.
- d. Migrating important data blocks, those that are highly accessed by users and take longer time to be decrypted, to the SSDs layer will reduce data retrieve latency. However, this is restricted by the I/O bandwidth available in the hybrid storage system.
- e. Other researchers have not provided in any means solutions that perform data migration in self-encryption hybrid storage system to improve data retrieve performance. They just provided performance evaluation of different encryption algorithms on such systems.
- f. The importance for having a simple complexity algorithm for migrating important data to the SSDs layer in such systems is significant.

In order to address the above mentioned research motivation, we introduce in this paper a linear complexity algorithm for migrating data between the hybrid storage system layers. The algorithm called hybrid self-encryption storage data migration (HSESM) keeps assessing the encrypted data blocks stored on the hybrid storage system to determine for each block if it has a significant importance for the users' applications to be placed on the SSDs layer. The model also takes into consideration the complexity of the decryption process in terms of time and if it has high significance of the total data block reading time. The algorithm then keeps adaptively moving the data blocks between the two storage layers by considering the available I/O bandwidth that is affected by the current use of the storage system. Up to our best knowledge, our research is the first that performs data migration for symmetric encrypted data based on the available I/O bandwidth in hybrid storage systems. Other researchers have examined the performance of different symmetric encryption algorithms using different storage devices [3].

The remaining parts of this paper as follow: section 2 illustrates the related work, section 3 proposes our HSESM algorithm, section 4 provides simulation and performance evaluation, and section 5 provides a research conclusion.

## 2. RELATED WORK

Here we discuss some topics related to our research including hybrid storage systems and symmetric self-encryption.

### 2.1. Hybrid storage systems

Data storage devices have developed during the past decades with different types of hardware. Magnetic HDDs are the oldest types of storage that are still used in big data centers as they can accommodate large amount of data with less cost per storage unit. SSDs were developed as a new type of storage devices that are much faster and energy efficient. They use flash NAND technology that has proven high read/write speed and low power consumption in comparison to HDDs [6]. As of their ability to handle huge volume of I/O requests per second (IOPS), they are still much expensive in comparison to HDDs and have not completely replaced them. Some research provided solutions to increase the number of I/O requests that can be processed in a second IOPS in non-volatile SSDs where it can reach to one million IOPS [7].

Hybrid storage systems use several layers of storage devices such as HDDs and SSDs. The important data that are expected to be accessed soon tend to be stored in the upper SSDs layer to facilitate

their access [1]. For large data centers, researchers suggest adding SSDs layer on the top of HDDs one to buffer users' frequently requested data block in the SSDs. Data blocks prefetching solutions were proposed to bring the data from HDDs to the SSDs before they are being actually requested by the users' applications. So, more data will be found in the SSDs [1]. As there exists high volume of I/O requests in such systems, [8] suggest using a small part of the SSDs in order to ease the scheduling process of the disk requests and to reduce the challenges on the HDDs storage.

Data centers tend to be parallel hybrid storage systems as they have multi-levels of storage devices where each level is in a form of an array. This provides high I/O bandwidth and helps in reading and writing many data blocks in parallel, perform data migration, and facilitating data prefetching. Data migration helps in moving hot data blocks that are highly requested for reads to the SSDs level [9]. Some research focused in reducing the unnecessary migration for data between the HDDs and the SSDs for the dying data to improve the performance of the hybrid storage system [10]. Many researchers proposed prefetching tools to leverage the available I/O bandwidth in parallel hybrid storage systems in order to make important data to be available in the SSDs level either by informed or predictive approach [11], [12].

## 2.2. Self-encryption systems and encryption algorithms

Self-encryption drives (SEDs) have hardware capabilities that enable them to perform the encryption and decryption process internally. Encryption engines that are integrated to the drivers' controllers are used for the cryptography process [11]. Cloud Data centers that use hybrid storage systems use SEDs system for data protection and security as they have many advantages when compared with software encryption. Advantages includes speed, less complexity high adoption, and device durability [13]. In addition, it uses many symmetric encryptions with different key sizes that do not need to be shared over the network.

Symmetric encryption algorithms perform the encryption/decryption process using a known key value that might be with different sizes and does not need to be shared over the network. There exist several symmetric encryption algorithms including advanced encryption standard (AES), data encryption standard (DES), triple data encryption standard (3DES), and Blowfish. They vary in performance especially in terms of encryption speed while using different data blocks sizes [14].

Many researchers suggest using hybrid encryption techniques that combine several encryption algorithms to leverage their capabilities and to improve the encryption strength [15]-[17]. Others have introduced hybrid encryption using chaotic system [18]. Asymmetric encryption algorithms such as Rivest-Shamir-Adleman (RSA) involve using a public and private key to secure the data transmission over the network. Symmetric encryption shows faster performance in terms of execution time in comparison to the asymmetric approach [19], [20]. Some algorithms used asymmetric encryption to protect the key of the symmetric encryption [21]. Symmetric encryption is more energy efficient and resilient for attacks based from quantum computers [22]. In study [23], authors have introduced a data security novel for hybrid cloud systems to ensure the protection of the user's data using three symmetric encryption approaches that are provided as-a-service. This indeed show the importance of such symmetric encryption solutions in cloud data centres. The authors of [24] have proposed a symmetric double encryption mechanism for securing the data sent and received in cloud systems in a form of two layers of cryptography. Some researcher found that there are some weaknesses that may exist in SSDs self-encryption and might be compromised due the specification and design issues. Hence, they recommend using more security measures and not only relying on hardware security [25].

For hybrid storage systems, researchers have only examined the performance of symmetric self-encryption algorithms performance on different layers. No research has investigated the possibilities of migrating self-encrypted data blocks among the different layers to leverage the high performance provided by the SSDs layer and to reduce the total data read latencies that include the decryption cost that might vary based on the complexity of the used algorithm and the key size.

## 3. PROPOSED SOLUTION

Hybrid storage systems consist of layers of HDDs and SSDs to provide variety of storage layers to store different types of data. They implement symmetric self-encryption mechanism to boost the security of the stored data. When requesting a data block that is already encrypted in such systems, the total read latency includes the time latency needed to have it decrypted. This latency may increase due to the complexity of the cryptography algorithm and the key size. Many sensitive data blocks require using more complex cryptography algorithms and extra long key sizes to increase their security level. This will increase the total read latency whenever a particular block is requested by the users' applications. As mentioned previously, the possibility of finding a data already placed in the HDDs layer is more due to their significant size. In

order to leverage the capabilities of hybrid storage systems, important data blocks have to be continuously assessed and migrated from the HDDs layer to the SSDs one in order to reduce the total data retrieve latency and the effect of the decryption process. This should take into consideration the available I/O bandwidth in the hybrid storage system that constrains the amount of data blocks that can be migrated among the two layers. HSESM algorithm is introduced to assess the important data blocks that are highly frequently accessed, belong to foreground processes, and show extended decryption latencies, in order to migrate them to the SSDs layer by leveraging the available I/O bandwidth of the storage system. Other research has only examined the performance of such symmetric algorithms on different types of storage systems [3].

In a hybrid storage system, there exists many I/O read requests that are continuously issued by users' applications for data blocks stored on the system during their execution. For each requested block, HSESM algorithm makes an assessment if a data block that is stored on the HDDs layer is important and should be moved to the SSDs one. Then it performs the migration process by taking into consideration the available I/O bandwidth between the two layers. In case the SSDs layer reached to its maximum capacity, the opposite should take place and the less accessed data blocks are moved to HDDs.

To efficiently perform the above-mentioned data migration process, HSESM algorithm consists of four threads that keep executing concurrently: i) assessment thread ( $T_{Assmt}$ ), ii) data migration from HDD to SSD thread ( $T_{Mig\_hdd\_ssd}$ ), iii) SSD full capacity assessment ( $T_{Full\_SSD}$ ), and iv) data migration from SSD to HDD thread ( $T_{Mig\_ssd\_hdd}$ ). Before their execution, there exists some global data structures and variables that are declared to facilitate the work of the threads.

### 3.1. Global data structures and variables

HSESM threads require the declaration for several global data structures and variables to function. There exists a list that contains information for each data block stored on the hybrid storage system ( $L_{B\_info}$ ). Each data block is allocated a node in this list and contains the following variables: ( $V_{Block\_number}$ ) to store a unique number for the data block, ( $V_{Access\_Freq}$ ) initialized to zero and contains the total number of times it was requested for reading, ( $V_{TDecpt}$ ) initialized to zero and contains the recent latency of time it took to be decrypted and it is usually in milliseconds, ( $V_{in\_SSD}$ ) Boolean variable where true value indicates that the block is allocated in the SSDs layer, ( $V_{Priority}$ ) Boolean variable where a true value indicates that the block belongs to a real-time or foreground application rather than a background one, ( $V_{SSDs\_scheduled}$ ) Boolean variable initialized to false where true value indicates that the block is scheduled for migration from the SSDs layer to the HDDs one, and ( $V_{HDDs\_scheduled}$ ) Boolean variable initialized to false where true value indicates that the block is scheduled for migration from the HDDs layer to the SSDs one.

In addition, there exists global variables declared to be used for calculations related to accessed data blocks. ( $V_{Hgst\_Freq}$ ) initialized to zero to store the highest recorded ( $V_{Access\_Freq}$ ) for the requested data blocks, ( $V_{Lst\_Freq}$ ) initialized to zero and to store the lowest recorded ( $V_{Access\_Freq}$ ) for the requested data blocks, ( $V_{Hgst\_TDecpt}$ ) initialized to zero to store the highest recorded ( $V_{TDecpt}$ ) for the requested data blocks, and ( $V_{Lst\_TDecpt}$ ) initialized to zero and to store the lowest recorded ( $V_{TDecpt}$ ) for the requested data blocks.

Let the global variable ( $V_{TotalSize}$ ) be the total size of the hybrid storage system in term of data blocks assuming fixed size data blocks will be stored on both layers. Let ( $V_{SSDsSize}$ ) be the SSDs layer size using the same fixed size data blocks. The ratio of the SSDs layer size in respect to the total size of the storage system is calculated in global variable ( $V_{R\_SSDs}$ ) using (1). Up to ( $V_{R\_SSDs}$ ) ratio of the entire data blocks will be stored on the SSDs layer.

$$V_{R\_SSDs} = V_{SSDsSize} \div V_{TotalSize} \quad (1)$$

A global queue structure ( $Q_{MigHDDs\_SSDs}$ ) is declared to queue the data blocks that are scheduled for migration from the HDDs layer to the SSDs one by enqueueing each ( $V_{Block\_number}$ ). ( $Q_{MigSSDs\_HDDs}$ ) is the same for the opposite direction.

In a hybrid storage system, there exists a maximum bandwidth value that represents the number of data blocks that can be read from the SSDs layer to the HDDs one and the opposite from the HDDs layer to the SSDs during the normal operation of the system while it is responding to the users I/O read and write requests [1], [5]. In case this number was exceeded, the read and writes operations on both layers will exceed their normal latencies. Let a global variable ( $V_{BW\_SSDsHDDs}$ ) be the maximum bandwidth for moving data from SSDs to the HDDs and a global variable ( $V_{BW\_HDDsSSDs}$ ) to be the one from the HDDs to the SSDs. The bandwidth increases in non-peak times as the pressure on the storage system decreases. Maximum bandwidth was measured by other research by sensing whenever the storage system reaches to an increased latency higher than an expected level during the process of performing the I/O requests [1]. In addition, most storage systems provide information on the maximum bandwidth in the specifications documents. In this algorithm, we will assume fixed values of maximum bandwidth in the performance evaluation section. However, the

algorithm is flexible for any variable values for the maximum bandwidth that can change over the execution time. Table 1 summarises the global data structures and variables.

Table 1. Summary of the global data structures and variables

Identifier	Description	Identifier	Description
$L_{B\_info}$	Unique data block number	$V_{Hgst\_TDecpt}$	Current highest value of $L_{B\_info}$ - $V_{TDecpt}$
$V_{Block\_number}$			
$L_{B\_info}$	Data block read requests	$V_{Lst\_TDecpt}$	Current lowest value of $L_{B\_info}$ - $V_{TDecpt}$
$V_{Access\_Freq}$			
$L_{B\_info}$ - $V_{TDecpt}$	Recent decryption latency in time	$V_{TotalSize}$	Total size of the storage system two layers in data blocks
$L_{B\_info}$ - $V_{in\_SSD}$	Is the block in SSDs	$V_{SSDsSize}$	SSDs layer size in data blocks
$L_{B\_info}$ - $V_{Priority}$	Real-time/foreground block	$V_{R\_SSDs}$	SSDs layer size ratio to the total size of the entire storage system
$L_{B\_info}$	The block scheduled for migration to SSDs	$Q_{MigHDDs\_SSDs}$	Queue of scheduled data blocks for migration to the SSDs
$V_{SSDs\_scheduled}$			
$L_{B\_info}$	The block scheduled for migration to HDDs	$Q_{MigSSDs\_HDDs}$	Queue of scheduled data blocks for migration to the HDDs
$V_{HDDs\_scheduled}$			
$V_{Hgst\_Freq}$	Current highest value of $L_{B\_info}$ - $V_{Access\_Freq}$ of the entire data blocks	$V_{BW\_SSDsHDDs}$	Maximum bandwidth of data block that can be moved from SSDs to the HDDs
$V_{Lst\_Freq}$	Current lowest value of $L_{B\_info}$ - $V_{Access\_Freq}$ of the entire data blocks	$V_{BW\_HDDsSSDs}$	Maximum bandwidth of data block that can be moved from HDDs to the SSDs

### 3.2. Assessment thread ( $T_{Assmt}$ )

This thread continuously evaluates each requested data block by the users' applications to decide if it is eligible for migration from the HDDs to the faster SSDs layer based on its importance. For each requested data block, it updates variables on how often it is accessed and how long it takes to have it decrypted, calculating these in global maximum and minimum variables. If the block is already in the SSDs layer, no further action takes place. For those requested data blocks allocated in the HDDs layer and belong to foreground or real-time processes, the thread calculates the percentiles for both, access frequency and decryption time latency based on the size of the SSDs layer relatively to the total size of the storage system. It also considers in this calculation the mentioned maximum and minimum values of the access frequencies and the decryption latencies. These percentiles are used to identify thresholds that indicate which blocks have the importance to be migrated to the SSDs. If a block's access frequency and decryption latency exceed these percentiles, it's scheduled for migration to the SSDs layer to improve its retrieval speed.

For each data block requested by the users' applications from the hybrid storage system regardless in which layer it is initially stored, this thread first retrieves its information node from the blocks information list ( $L_{B\_info}$ ). First, it increments its access frequency variable ( $V_{Access\_Freq}$ ). Then, it updates the two global variables: highest access frequency ( $V_{Hgst\_Freq}$ ) and lowest access frequency ( $V_{Lst\_Freq}$ ). Their update is a simple comparison, where if the updated value ( $V_{Access\_Freq}$ ) of the block is more than or equals ( $V_{Hgst\_Freq}$ ), it will replace its value. Same thing happens if it is less than or equals ( $V_{Lst\_Freq}$ ). At the same time, the thread calculates the latency that the block took to be self-decrypted using its symmetric encryption algorithm, and store that value in ( $V_{TDecpt}$ ). The calculation of the self-decryption process latency is done by simple timer. Then, maximum and minimum decryption latency global variables ( $V_{Hgst\_TDecpt}$  and  $V_{Lst\_TDecpt}$  respectively) are updated similarly to the approach done with access frequency variables.

At this point, the thread reads the values ( $V_{in\_SSDs}$ ), ( $V_{SSDs\_scheduled}$ ), and ( $V_{Priority}$ ) of the data block. ( $V_{in\_SSDs}$ ) indicates if the block is already allocated in the SSDs layer, ( $V_{SSDs\_scheduled}$ ) indicated if it was already scheduled for migration to the SSDs layer, and ( $V_{Priority}$ ) indicates if the block belongs to a real-time or foreground application. In case both ( $V_{in\_SSDs}$ ) or ( $V_{SSDs\_scheduled}$ ) were true or ( $V_{Priority}$ ) was false, the thread will not continue its remaining part on the current block and waits for the next requested one as the current data block is either already scheduled for migration to the SSDs layer or it is already allocated over there, or it belongs to a background application where its reading speed efficiency is not an issue. Otherwise, the thread continues to the next steps as the data block might be eligible for migration since it is allocated in the HDDs layer, not scheduled for migration, and belongs to a priority application.

Based on the ratio of the SSDs layer size in respect to the total size of the storage system ( $R_{SSDs}$ ), the percentile ( $P_{Access\_Freq}$ ) is calculated using (2) that represents the minimum value of ( $V_{Access\_Freq}$ ) where data blocks who have at least this value are important be migrated to the SSDs layer. This is assuming that the data blocks access frequency values ( $V_{Access\_Freq}$ ) are uniformly distributed. This assumption is valid as in any system, there exists many data blocks that are much frequently accessed, others that are rarely accessed, and many others that are accessed with normal frequencies.

$$P_{Access\_Freq} = V_{Lst\_Freq} + V_{R\_SSDs} \times (V_{Hgst\_Freq} - V_{Lst\_Freq}) \quad (2)$$

Similar to (2), the percentile ( $P_{TDecpt}$ ) is calculated using (3) that represents the minimum value of ( $V_{TDecpt}$ ) where data blocks who have at least this value are important to be migrated to the SSDs layer as they have the highest decryption latency. So, finding them in the SSDs layer is more important to speed up their reading process. We make here the same assumption where decryption time values of the data blocks ( $V_{TDecpt}$ ) are uniformly distributed.

$$P_{TDecpt} = V_{Lst\_TDecpt} + V_{R\_SSDs} \times (V_{Hgst\_TDecpt} - V_{Lst\_TDecpt}) \quad (3)$$

After the calculation of the two percentiles local variables, the thread compares if the data block access frequency ( $V_{Access\_Freq}$ ) is greater than or equals the percentile ( $P_{Access\_Freq}$ ) and if the decryption time of the data block ( $V_{TDecpt}$ ) was more than or equals the percentile ( $P_{TDecpt}$ ). In this case, the data block is considered an important block that must be migrated to the SSDs layer to have the latencies of all of its future read requests minimized. The block number ( $V_{Block\_number}$ ) is enqueued to the end of the queue ( $Q_{MigHDDs\_SSDs}$ ) so it is scheduled to be moved to the SSDs layer by the thread ( $T_{Mig\_hdd\_ssd}$ ) whenever a slot of the storage system bandwidth allows. The variable ( $V_{SSDs\_scheduled}$ ) is set to true so if it was requested again by the user's application before it is migrated to the SSDs layer, the thread will not add it again to the queue. Algorithm 1 shows a pseudocode for ( $T_{Assmt}$ ) thread that was explained.

#### Algorithm 1. ( $T_{Assmt}$ ) pseudocode

```

while (true)
    for each accessed data block do
        retrieve ( $L_B\_info$ ) node
        increment ( $V_{Access\_Freq}$ )
        update ( $V_{Hgst\_Freq}$ ) and ( $V_{Lst\_Freq}$ ) based on ( $V_{Access\_Freq}$ )
        calculate ( $V_{TDecpt}$ ) self-decryption latency
        update ( $V_{Hgst\_TDecpt}$ ) and ( $V_{Lst\_TDecpt}$ ) based on ( $V_{TDecpt}$ )
        if ( $V_{in\_SSDs} == False$  and  $V_{SSDs\_scheduled} == False$  and  $V_{Priority} == True$ )
            calculate ( $P_{Access\_Freq}$ ) based on (2)
            calculate ( $P_{TDecpt}$ ) based on (3)
            if ( $V_{Access\_Freq} \geq P_{Access\_Freq}$  and  $V_{TDecpt} \geq P_{TDecpt}$ )
                enqueue ( $V_{Block\_number}$ ) to the end of ( $Q_{MigHDDs\_SSDs}$ )
                set ( $V_{SSDs\_scheduled}$ ) to True
            end if
        end if
    end for
end while

```

### 3.3. Data migration from hard disk drive to solid-state drive thread ( $T_{Mig\_hdd\_ssd}$ )

This thread performs the migration process for important data blocks to the SSDs layer that were determined by ( $T_{Assmt}$ ) thread. It keeps monitoring the current number of ongoing I/O reads from the HDDs and I/O writes on the SSDs to determine if the value ( $V_{BW\_HDDsSSDs}$ ) was not reached. The number of these ongoing I/O requests are considered a consumption of the bandwidth, and it is recorded in a local variable ( $V_{BCon\_HDDs\_SSDs}$ ). The HDDs to SSDs migration size that represents the number of data blocks scheduled for migration in ( $Q_{MigHDDs\_SSDs}$ ) and can be moved immediately based on the available bandwidth is calculated by (4), stored in local variable ( $V_{MigSize\_HDDsSSDs}$ ).

$$V_{MigSize\_HDDsSSDs} = V_{BW\_HDDsSSDs} - V_{BCon\_HDDs\_SSDs} \quad (4)$$

Then, the first ( $V_{MigSize\_HDDsSSDs}$ ) count of data blocks in the front of ( $Q_{MigHDDs\_SSDs}$ ) will be moved from the HDDs layer to the SSDs one. There nodes in ( $Q_{MigHDDs\_SSDs}$ ) will be dequeued. In addition, each block ( $V_{in\_SSDs}$ ) and ( $V_{SSDs\_scheduled}$ ) variables in ( $L_B\_info$ ) are set to True and False respectively. Algorithm 2 shows a pseudocode for ( $T_{Mig\_hdd\_ssd}$ ) thread that was explained.

#### Algorithm 2. ( $T_{Mig\_hdd\_ssd}$ ) pseudocode

```

while (true)
     $V_{BCon\_HDDs\_SSDs}$  = ongoing HDDs read and SSDs write requests
    if  $V_{BCon\_HDDs\_SSDs} < V_{BW\_HDDsSSDs}$ 
        calculate  $V_{MigSize\_HDDsSSDs}$  using (4)
        for each of the first  $V_{MigSize\_HDDsSSDs}$  nodes in  $Q_{MigHDDs\_SSDs}$ 
            copy block number  $V_{Block\_number}$  to SSDs Layer
            delete block number  $V_{Block\_number}$  from HDDs Layer
            update block  $V_{in\_SSDs}$  in  $L_B\_info$  to True
            update block  $V_{SSDs\_scheduled}$  in  $L_B\_info$  to False
        end for
    end if
end while

```

```

        dequeue block node from QMigHDDs_SSDs
    end for
end if
end while

```

### 3.4. Solid-state drives full capacity assessment ( $T_{Full\_SSD}$ )

This thread keeps calculating the number of data blocks written to SSDs layer to check if it has reached its maximum capacity ( $V_{SSDsSize}$ ) especially that SSDs layer is significantly less in size than the HDDs one. It uses a local variable ( $V_{SSD\_Blocks\_Count}$ ) to calculate the number of data blocks in the SSDs. To initialize the variable, the thread goes through ( $L_{B\_info}$ ) for one time and increments ( $V_{SSD\_Blocks\_Count}$ ) whenever ( $V_{in\_SSDs}$ ) is found true. The thread then keeps monitoring whenever a block is written on the SSDs layer to increment ( $V_{SSD\_Blocks\_Count}$ ). If the maximum size is reached, a data block in the SSDs layer must be scheduled for migration from the SSDs to the HDDs. The least complex approach to perform that is to linearly go through ( $L_{B\_info}$ ) list and to keep finding the data blocks that are allocated in the SSDs layer where ( $V_{in\_SSD}$ ) is true and have value of access frequency ( $V_{Access\_Freq}$ ) that became less than the percentile ( $P_{Access\_Freq}$ ) which is calculated here using (2). As percentile ( $P_{Access\_Freq}$ ) keeps changing while data blocks are accessed, there must be data blocks in the SSDs layer that their access frequency become less than the percentile and become less important. Hence, these blocks will be moved to the HDDs layer to open a space for new important data blocks. Whenever a block is found, it ( $V_{Block\_number}$ ) will be enqueued to the end of ( $Q_{MigSSDs\_HDDs}$ ) and the value of ( $V_{SSD\_Blocks\_Count}$ ) is decremented. The block ( $V_{HDDs\_scheduled}$ ) is set to true so it will not be rescheduled again before its migration. Algorithm 3 shows a pseudocode for ( $T_{Full\_SSD}$ ) thread that was explained.

#### Algorithm 3. ( $T_{Full\_SSD}$ ) pseudocode

```

for all blocks in  $L_{B\_info}$  do
    if  $V_{in\_SSDs} == True$ 
        increment  $V_{SSD\_Blocks\_Count}$ 
    end if
end for
while (true)
    if a new block written on the SSDs Layer
        increment  $V_{SSD\_Blocks\_Count}$ 
        calculate percentile  $P_{Access\_Freq}$  using (2)

        if  $V_{SSD\_Blocks\_Count} == V_{SSDsSize}$ 
            for each node in  $L_{B\_info}$ 
                if  $V_{in\_SSD} == True$  and  $V_{HDDs\_scheduled} == False$ 
                    if  $V_{Access\_Freq} < P_{Access\_Freq}$ 
                        enqueue  $V_{Block\_number}$  to the end of  $Q_{MigSSDs\_HDDs}$ 
                        decrement  $V_{SSD\_Blocks\_Count}$ 
                        set  $V_{HDDs\_scheduled}$  to True
                    end if
                end if
            end for
        end if
    end if
end while

```

### 3.5. Data migration from solid-state drive to hard disk drive thread ( $T_{Mig\_ssd\_hdd}$ )

This thread performs the migration of the data blocks to the HDDs layer whenever the SSDs are full for those data block determined by thread ( $T_{Full\_SSD}$ ). Similar to ( $T_{Mig\_hdd\_ssd}$ ), it keeps monitoring the current number of ongoing I/O reads from the SSDs and I/O writes on the HDDs to determine if the value of the bandwidth ( $V_{BW\_SSDsHDDs}$ ) was not reached. It also records the consumption of the bandwidth in a local variable ( $V_{BCon\_SSDs\_HDDs}$ ). Similarly, the size of SSDs to HDDs migration that represents the number of data blocks scheduled for migration in ( $Q_{MigSSDs\_HDDs}$ ) and can be moved immediately based on the available bandwidth is calculated by (5) in a local variable ( $V_{MigSize\_SSDsHDDs}$ ).

$$V_{MigSize\_SSDsHDDs} = V_{BW\_SSDsHDDs} - V_{BCon\_SSDs\_HDDs} \quad (5)$$

Then, the first ( $V_{MigSize\_SSDsHDDs}$ ) count of data blocks in the front of ( $Q_{MigSSDs\_HDDs}$ ) will be moved from the SSDs layer to the HDDs one. There nodes in ( $Q_{MigSSDs\_HDDs}$ ) will be dequeued. In addition, each block ( $V_{in\_SSDs}$ ) and ( $V_{HDDs\_scheduled}$ ) variables in ( $L_{B\_info}$ ) are both set to False. Algorithm 4 shows a pseudocode for ( $T_{Mig\_ssd\_hdd}$ ) thread.

**Algorithm 4. ( $T_{\text{Mig\_ssd\_hdd}}$ ) pseudocode**

```

while (true)
     $V_{\text{BCon\_SSDs\_HDDs}}$  = ongoing SSDs read and HDDs write requests
    if  $V_{\text{BCon\_SSDs\_HDDs}} < V_{\text{BW\_SSDsHDDs}}$ 
        calculate  $V_{\text{MigSize\_SSDsHDDs}}$  using (5)

        for each of the first  $V_{\text{MigSize\_SSDsHDDs}}$  nodes in  $Q_{\text{MigSSDs\_HDDs}}$ 
            copy block number  $V_{\text{Block\_number}}$  to HDDs Layer
            delete block number  $V_{\text{Block\_number}}$  from SSDs Layer
            update block  $V_{\text{in\_SSDs}}$  in  $L_{\text{B\_info}}$  to False
            update block  $V_{\text{HDDs\_scheduled}}$  in  $L_{\text{B\_info}}$  to False
            dequeue block node from  $Q_{\text{MigSSDs\_HDDs}}$ 
        end for
    end if
end while

```

**3.6. Hybrid self-encryption storage data migration algorithm complexity analysis**

As mentioned previously, HSESM to our best knowledge and review to the literature is the only algorithm that performs data migration for important data blocks in self-encryption hybrid storage systems. Analysing its complexity is important to show if it will be able to achieve its goals with minimal steps and time. The thread ( $T_{\text{Assmt}}$ ) in each iteration evaluates ( $n$ ) blocks access frequency and decryption latency and updates the global variables. Hence, it has a time complexity of  $O(n)$  per iteration in the worst-case scenario. ( $T_{\text{Full\_SSD}}$ ) thread also has a complexity of  $O(n)$  in the worst-case scenario when assessing all the data blocks in the SSDs layer. The two data migration threads ( $T_{\text{Mig\_hdd\_ssd}}$  and  $T_{\text{Mig\_ssd\_hdd}}$ ) move data blocks between the two layers based on the available I/O bandwidth. In case ( $k$ ) data blocks will be moved from the HDDs to the SSDs and ( $m$ ) data blocks will be moved in the opposite direction, the complexity of the threads per execution cycle will be  $O(k)$  and  $O(m)$  respectively considering the worst-case scenario. In fact, the size of ( $m$ ) is expected to be less than the size of ( $k$ ) due to the increased volume of the HDDs layer. The overall algorithm's complexity with its four threads hence will be in  $O(n)$ . This indicates that the algorithm is computationally light, quick to execute, and highly scalable. Figure 1 shows the algorithm complexity asymptotic notation per thread.

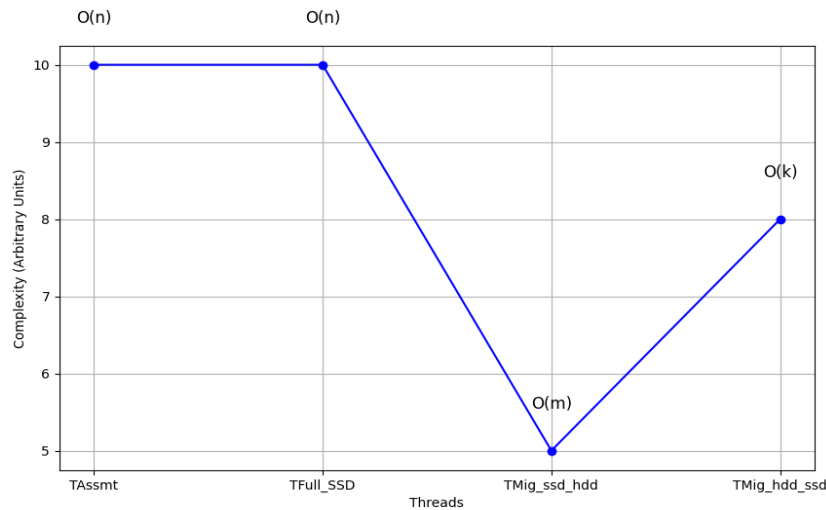


Figure 1. Algorithm complexity asymptotic notation per thread, overall complexity:  $O(n)$

**4. PERFORMANCE EVALUATION**

Due to the migration process of the important data blocks that HSESM perform to the SSDs by using the available I/O bandwidth, it leverages the high speed of the SSDs layer and reduces the data retrieve time in a significant amount. This improves the performance of users' applications. In this section, we present a simulation for the HSESM to evaluate its performance benefits that can be achieved in hybrid storage systems implementing symmetric self-encryption.

The performance metric that is used to measure the performance of HSESM algorithm is the total read latency for the entire data blocks that were requested by the users' applications during their execution over a particular period of time. The comparison will be in the case of using HSESM Algorithm with the case



of not putting it in use. The difference ratio represents how much the algorithm contributed in optimizing data retrieve performance of the hybrid data center that implements symmetric self-encryption. Whenever more requested data blocks are founded in the SSDs layer, their total reading latency will be less under the presence of decryption process latency.

For that purpose, we built a simulator using python that implements HSESM algorithm and execute it on a randomly generated trace that reflects real-world scenarios for records of users' applications requests for data blocks. There exist many factors that affects the performance evaluation of the algorithm. First, the locality of reference where if the users' applications read request tend to be more for the data blocks that are initially in the SSDs layer, hence, less migration will be needed. Second, the size of the SSDs layer in comparison to the HDDs one and the initial placement of the data blocks, indeed less SSDs size will decrease the amount of data blocks founded in the SSDs layer. Next, the ratio of the requested data blocks in the HDDs layer that were classified as important blocks and need to be migrated to the SSDs layer during a particular period of the system execution time. More important blocks will lead to extra need for data blocks to be migrated. In addition, the available bandwidth for moving the data blocks from the HDDs layer to the SSDs one whenever a block is scheduled for migration and the opposite whenever the SSDs layer becomes full in capacity. More consumed bandwidth in both directions will reduce the speed of the migration process and the chance of finding more data blocks in the SSDs layer. Both the randomly generated trace and the simulator consider different scenarios of the mentioned factors. The trace consists of 2000 records of users' applications read requests, where we found this number is enough to simulate our algorithm and to prove its performance over enough duration of time.

We assume a conservative assumption that the size of the SSDs layer is 30% of the total hybrid storage system and the size of the HDDs one is 70%. This is due to reasons that we mentioned previously. For data reference locality, we can simulate different scenario, due to the space limitation, we simulated a worst-case scenario when only 30% of the requested data blocks in the trace are initially in the SSDs layer.

Similar to Assaf *et al.* [5], we set the size of each data block stored in the hybrid storage system to 10 MB for the purpose of performance evaluation as data centers tend to use relatively large data blocks to improve the systems performance. However, our solution is flexible under the case of using different sizes of data blocks.

We implemented the following symmetric self-encryption algorithms for encrypting the data blocks in the system: AES-128, AES-256, Blowfish-128, Blowfish-256, and 3DES. Table 2 shows the range of decryption latencies in seconds that we got while decrypting the data blocks each of size 10 MB during the simulation using a normal speed processor (1.6 GHz Dual-Core Intel Core i5). In fact, the choice of the encryption algorithm is related to the application and the nature of the sensitive data need to be protected.

Table 2. Ranges of decryption latencies for AES, blowfish, and 3DES for data blocks of size 10 MB

Decryption algorithm	Latency range in (s)
AES-128	0.035465-0.058096
AES-256	0.039253-0.049405
Blowfish-128	0.108130-0.142406
Blowfish-256	0.120671-0.149756
3DES	0.474093-0.630230

Regardless of processing unit used in the self-encryption system, both AES and Blowfish show increased decryption latency when using larger key size. 3DES shows more complexity and high decryption latency in comparison to the others. Hence, data blocks that tend to be frequently used, their decryption latency is high, and belongs to real-time or foreground applications will be classified as important blocks that are subject for migration to the SSDs layer in case they exceed the updated value of the percentiles calculated in (2) and (3).

For the ratio of the important data blocks, we programmed our trace generator to take different scenarios of the amount of data blocks to be classified as important. We simulated three scenarios were 50%, 40%, and 30% of the trace requested data blocks that are initially in the HDDs layer are classified as important. We assume that 50% is a good upper limit for that ratio due to the increased volume of the HDDs layer as well as for the fact that most of the important data blocks tend to be initially allocated in SSDs layer.

For the bandwidth available for migrating the data between the two layers, we assume a conservative assumption that it will reach to 70% of consumption during the peak times and then it relaxes to 60% and 50% when there becomes less pressure on the storage system. Higher consumption rate of the available bandwidth will reduce the system efficiency and raise the need for scalability.

The final input needed in the simulation is the disk read latencies in seconds from the HDDs ( $\text{Latency}_{\text{HDDs}}$ ) and SSDs ( $\text{Latency}_{\text{SSDs}}$ ) as well as the latency in seconds for reading a data block from the HDDs and have it written in the SSDs or the opposite direction ( $\text{Latency}_{\text{HDDs-SSDs}}$ ) when using a block size of 10 MB. We used the validated values in [5] where the ranges of these values are illustrated in Table 3.

Table 3. Validated values of disk latencies parameters

Disk latency parameter	Latency range in (s)
$\text{Latency}_{\text{HDDs}}$	0.075-0.12
$\text{Latency}_{\text{SSDs}}$	0.045-0.052
$\text{Latency}_{\text{HDDs-SSDs}}$	0.115-0.122

After executing the simulator using the different scenarios, we found that the total disk read latency without using HSESM is (199.1s) as the locality of the requested data blocks is set to be 30% in the SSDs. Implementing HSESM will always result reduced total disk read latencies under different scenarios mentioned above of consumed bandwidth and the amount of data blocks being classified as important. Figure 2 shows that HSESM data migration process has contributed in improving the performance of the hybrid storage system between 13.71% and 23.61% of total disk read latency reduction. HSESM also has contributed in reducing the effect of the decryption latency for those data blocks frequently requested by the users' applications especially when they tend to use more complex symmetric encryption algorithms with larger key sizes. It clearly shows that whenever the consumed bandwidth is less and the number of the encrypted data blocks that were classified as important are more, the ratio of improvement is higher. For some short period of time, the case of bandwidth consumption of 60% shows less improvement ratio when 50% of data blocks were classified as important. This was due to the increased opposite migration to the HDDs that took place in that period during the simulation. But in general, the curve follows the mentioned general performance trend. As we mentioned in the introduction that after thorough research, our research is the first that provides data migration solution for important symmetric encrypted data blocks in hybrid storage systems. Hence, our results and findings can offer a valuable baseline for any potential similar future research.

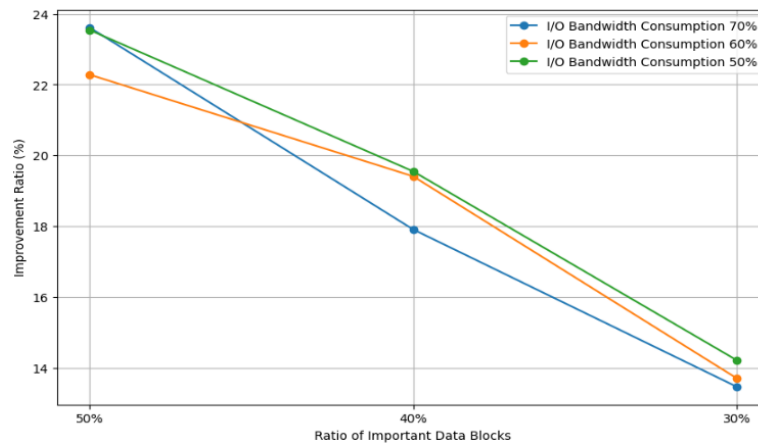


Figure 2. Improvement ratio of total disk read latency in (%) provided by HSESM under different scenarios of the amount of important data blocks and the consumed I/O bandwidth for data migration

## 5. CONCLUSION

In this paper, we introduced a data migration algorithm (HSESM) for symmetric self-encryption data centers that implement hybrid storage systems of SSDs and HDDs layers. It keeps migrating frequently used data blocks that belong to real-time or foreground processes and require more decryption latency to the SSDs layer in order to take advantage of its increased speed performance. The algorithm leverages the available I/O bandwidth of the storage system to perform the migration process aiming to decrease the total data read latencies. HSESM was simulated under different scenarios when the data blocks were encrypted using AES, blowfish, 3DES with different key sizes. It was able to decrease the total read latency of the requested data blocks by the users' applications in 13.71%-23.61%. One important future work that we would like to investigate is the effect of data migration on power consumption in self-encryption hybrid storage systems considering the energy efficiency of SSDs layer.

## FUNDING INFORMATION

Authors state no funding involved.

## AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Maen M. Al Assaf	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	
Mohammad Qatawneh		✓		✓	✓	✓	✓			✓				
AlaaAldin AlRadhi	✓		✓			✓		✓		✓	✓			

C : Conceptualization

M : Methodology

So : Software

Va : Validation

Fo : Formal analysis

I : Investigation

R : Resources

D : Data Curation

O : Writing - Original Draft

E : Writing - Review & Editing

Vi : Visualization

Su : Supervision

P : Project administration

Fu : Funding acquisition

## CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

## DATA AVAILABILITY

The data that support the findings of this study were generated and used through a python based simulation that was carefully built to reflect real-world scenarios. No external data sets were used in this research. The data that support the findings of this study are available from the corresponding author, [AA, MA, MQ], upon reasonable request.




## REFERENCES

- [1] M. M. Al Assaf, X. Jiang, X. Qin, M. R. Abid, M. Qiu, and J. Zhang, "Informed Prefetching in Distributed Multi-Level Storage Systems," *Journal of Signal Processing Systems*, vol. 90, pp. 619–640, 2018, doi: 10.1007/s11265-017-1277-z.
- [2] A. F. Mhdawy and M. M. Al Assaf, "An Energy Efficient Approach for Big Data Mass Storage Systems Using A Sequential Cache," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 23, pp. 6882-6890, 2022.
- [3] B. A. Sassani (Sarrafpour), M. Alkorbi, N. Jamil, M. A. Naeem, and F. Mirza, "Evaluating Encryption Algorithms for Sensitive Data Using Different Storage Devices," *Scientific Programming*, vol. 2020, pp. 1-9, 2020, doi: 10.1155/2020/6132312.
- [4] J. Kim *et al.*, "Self-Encrypting Drive Evolving Toward Multitenant Cloud Computing," *Computer*, vol. 57, no. 2, pp. 79-90, Feb. 2024, doi: 10.1109/MC.2023.3308955.
- [5] M. M. Al Assaf, X. Jiang, M. R. Abid, and X. Qin, "Eco-Storage: A Hybrid Storage System with Energy-Efficient Informed Prefetching," *Journal of Signal Processing Systems*, vol. 72, no. 3, pp. 165-180, 2013, doi: 10.1007/s11265-013-0784-9.
- [6] H. Riggs, S. Tufail, I. Parvez, and A. Sarwat, "Survey of Solid State Drives, Characteristics, Technology, and Applications," *2020 SoutheastCon*, 2020, pp. 1-6, doi: 10.1109/SoutheastCon44009.2020.9249760.
- [7] J. Zhang, M. Kwon, M. Swift, and M. Jung, "Manycore-based scalable SSD architecture towards one and more million IOPS," *Annual Non-Volatile Memories Workshop (NVMW)*, 2021.
- [8] X. Zhang, K. Davis, and S. Jiang, "iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O," *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 715-726, doi: 10.1109/IPDPS.2012.70.
- [9] J. Niu, J. Xu and L. Xie, "Hybrid Storage Systems: A Survey of Architectures and Algorithms," in *IEEE Access*, vol. 6, pp. 13385-13406, 2018, doi: 10.1109/ACCESS.2018.2803302.
- [10] M. Lin, R. Chen, J. Xiong, X. Li, and Z. Yao, "Efficient Sequential Data Migration Scheme Considering Dying Data for HDD/SSD Hybrid Storage Systems," in *IEEE Access*, vol. 5, pp. 23366-23373, 2017, doi: 10.1109/ACCESS.2017.2766667.
- [11] M. M. Al Assaf, A. Rodan, M. Qatawneh, and M. R. Abid, "A Comparison Study between Informed and Predictive Prefetching Mechanisms for I/O Storage Systems," *International Journal of Communications, Network and System Sciences*, vol. 8, no. 5, pp. 181-186, 2015, doi: 10.4236/ijcns.2015.85019.
- [12] M. Al Assaf, "Performance Optimization for Distributed Hybrid Storage Systems Using a Predictive Approach," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 4, pp. 4819-4826, 2020, doi: 10.30534/ijatse/2020/91942020.
- [13] T. Coughlin, "Solid security: The rise of self-encrypting solid state drives," *SNIA (Solid State Storage Initiative)*, 2011.
- [14] M. N. Alenezi, H. Alabdulrazzaq, and N. Q. Mohammad, "Symmetric encryption algorithms: Review and evaluation study," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 12, no. 2, pp. 256-272, 2020.
- [15] P. Kuppuswamy, S. Q. Y. A. K. Al-Maliki, R. John, M. Haseebuddin, and A. A. S. Meeran, "A hybrid encryption system for communication and financial transactions using RSA and a novel symmetric key algorithm," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 12, no. 2, pp. 1148-1158, 2023, doi: 10.11591/eei.v12i2.4967.
- [16] P. Bharathi, G. Annam, J. B. Kandhi, V. K. Duggana, and A. T., "Secure File Storage using Hybrid Cryptography," *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, 2021, pp. 1-6, doi: 10.1109/ICCES51350.2021.9489026.




- [17] V. Sharma, A. Chauhan, H. Saxena, S. Mishra and S. Bansal, "Secure File Storage on Cloud using Hybrid Cryptography," *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, 2021, pp. 1-6, doi: 10.1109/ISCON52037.2021.9702323.
- [18] M. Hamdi, J. Miri, and B. Moalla, "Hybrid encryption algorithm (HEA) based on chaotic system," *Soft Computing*, vol. 25, pp. 1847-1858, 2021, doi: 10.1007/s00500-020-05258-z.
- [19] R. B. Marqas, S. M. Almufti, and R. R. Ihsan, "Comparing Symmetric and Asymmetric cryptography in message encryption and decryption by using AES and RSA algorithms," *Journal of Xi'an University of Architecture & Technology*, vol. 12, no. 3, pp. 3110-3116, 2020.
- [20] M. A. Al-Shabi, "A survey on symmetric and asymmetric cryptography algorithms in information security," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, no. 3, pp. 576-589, 2019, doi: 10.29322/IJSRP.9.03.2019.p8779.
- [21] Q. Zhang, "An Overview and Analysis of Hybrid Encryption: The Combination of Symmetric Encryption and Asymmetric Encryption," *2021 2nd International Conference on Computing and Data Science (CDS)*, 2021, pp. 616-622, doi: 10.1109/CDS52072.2021.00111.
- [22] B. Halak, Y. Yilmaz and D. Shiu, "Comparative Analysis of Energy Costs of Asymmetric vs Symmetric Encryption-Based Security Applications," in *IEEE Access*, vol. 10, pp. 76707-76719, 2022, doi: 10.1109/ACCESS.2022.3192970.
- [23] A. A. Fairrosebanu and A. C. N. Jebaseeli, "Data security in cloud environment using cryptographic mechanism," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 12, pp. 462-471, 2023 doi: 10.11591/eei.v12i1.4590.
- [24] M. Nadeem, A. Arshad, S. Riaz, S. W. Zahra, S. S. Band, and A. Mosavi, "Two Layer Symmetric Cryptography Algorithm for Protecting Data from Attacks," *Computers, Materials & Continua*, vol. 74, no. 2, 2023, doi: 10.32604/cmc.2023.030899.
- [25] C. Meijer and B. van Gastel, "Self-Encrypting Deception: Weaknesses in the Encryption of Solid State Drives," *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 72-87, doi: 10.1109/SP.2019.00088.

## BIOGRAPHIES OF AUTHORS






**Maen M. Al Assaf**    is an Associate Professor of Computer Science at the University of Jordan. He received his Ph.D. in Computer Science from Auburn University in the State of Alabama, USA in 2011. He published several research works in Distributed Storage Systems. His research interests are in many fields including distributed systems, operating systems, cloud computing, IoT systems, and edge computing. He can be contacted at email: m\_lassaf@ju.edu.jo.



**Mohammad Qatawneh**    is a Professor of Computer Science at the University of Jordan and Al-Ahliyya Amman University. He received his Ph.D in Computer Engineering from Kiev University in 1996. He published several papers in the areas of parallel algorithms, networks and embedding systems. His research interests include blockchain, cybersecurity, IoT, and digital forensics. He can be contacted at email: mohd.qat@ju.edu.jo.



**AlaaAldin AlRadhi**    is a Professor at Sheridan College, in Toronto, Ontario, Canada. He received his Master's degree in Computer Information and Network Security from DePaul University in 2008. His research and teaching interests are in IPv6, cyber security, AWS/Azure/GCP clouds, and AI/data science. He is an IPv6 certified trainer and administrator (gold). Recently he received many awards including Seneca college "research influencer" award and Sheridan college "teaching character award" 2022. He also was nominated for Ontario Minister of Colleges/Universities "awards of excellence". He can be contacted at email: Alaaaldin.alradh@sheridancollege.ca.